DOCUMENT RESUME

ED 477 463 TM 034 993

AUTHOR	Scandura, Joseph M.
TITLE	Structural Learning Theory: Current Status and New Perspectives.
PUB DATE	2003-04-00
NOTE	32p.
AVAILABLE FROM	MERGE Research Institute, 12496 Greentree Lane, Narberth, PA 19072.
PUB TYPE	Reports - Descriptive (141)
EDRS PRICE	EDRS Price MF01 Plus Postage. PC Not Available from EDRS
DESCRIPTORS	Child Development; Educational Research; *Educational Theories; *Instruction
IDENTIFIERS	*Structural Learning

ABSTRACT

This paper discusses the current status of and new perspectives on the Structural Learning Theory (SLT). Special consideration is given to how SLT has been influenced by recent research in software engineering, and the range of possibilities it opens for instructional research and practice in the 21st century. Starting with fundamental precepts of the instructional process, a generalization of the SLT is proposed that offers an integrated, parsimonious, operational and predictive, as well as explanatory, account of competence, cognition, and behavior potentially from birth onward, and their implications for instruction. Supporting examples and experimental research are cited in context. (Contains 6 figures and 48 references.) (Author/SLD)

Reproductions supplied by EDRS are the best that can be made
from the original document.

ED 477 463

Structural Learning Theory: Current Status and New Perspectives

Joseph M. Scandura

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL IN MICROFICHE, AND IN ELECTRONIC MEDIA FOR ERIC COLLECTION SUBSCRIBERS ONLY, HAS BEEN GRANTED BY

A. B. Scandura

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

2A

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☑ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

TM034993

BEST COPY AVAILABLE

Structural Learning Theory: Current Status and New Perspectives. [1]

Joseph M. Scandura
MERGE Research Institute, Scandura.com and University of Pennsylvania

ABSTRACT

This article presents the current status and new perspectives on the Structural Learning Theory (SLT). Special consideration is given to how SLT has been influenced by recent research in software engineering, and the range of possibilities it opens for instructional research and practice in the twenty-first century. Starting with fundamental precepts of the instructional process, a generalization of the SLT is proposed that offers an integrated, parsimonious, operational and predictive (as well as explanatory) account of competence, cognition and behavior potentially from birth onward, and their implications for instruction. Supporting examples and experimental research are cited in context.

Key Words
Cognition, Constructivism, Piaget, Instructional Theory, Instructional Technology, Instructional Design, Task Analysis, Structural Learning Theory

INTRODUCTION

This article presents the current status and new perspectives on the Structural Learning Theory (SLT)[2]. Special consideration is given to how SLT has been influenced by recent research in software engineering, and the range of possibilities it opens for future instructional research and practice. Starting with fundamental precepts of the instructional process, a generalization of the SLT is proposed that offers an integrated, parsimonious, operational and predictive (as well as explanatory) account of competence, cognition and behavior potentially from birth onward, and their implications for instruction. Supporting examples and experimental research are cited in context.

OVERVIEW

Instructional systems are conceptualized in terms of interactions between a tutor and a learner (more generally between any two or more individuals) with respect some shared content domain (Figure 1). The basic question is how knowledge changes over time as a result of these interactions.

SLT attempts to answer the following basic questions and derives from the associated basic assumptions:

---

[1] The term "structural" in the title refers to the SLT's original emphasis on complex structured subject matters such as mathematics.

[2] The SLT evolved over a period of several decades beginning in the 1960's. Appendix A provides a useful overview of major developments and related publications over the years.

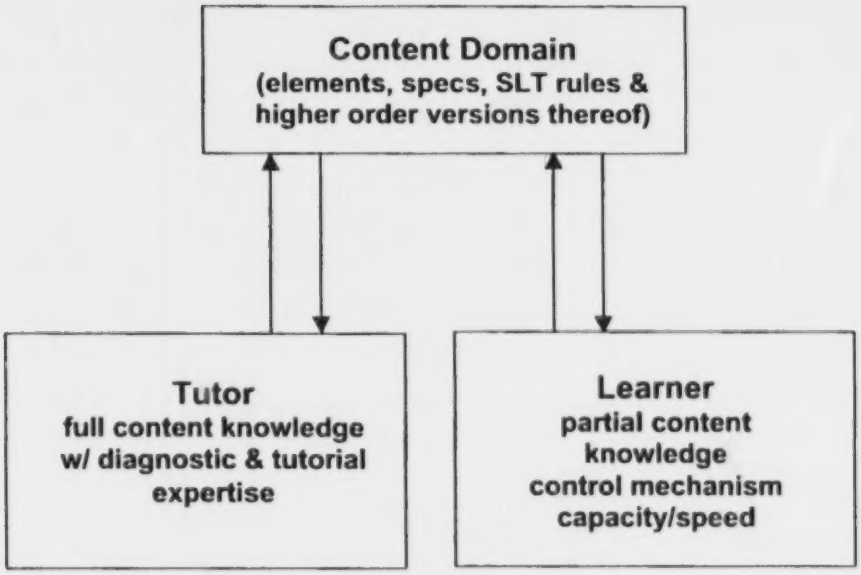

Figure 1. Key elements and constructs in the instructional process.

1. What does it mean to know something, and how can that competence be represented in a way that has behavioral relevance? Exactly what is observable behavior in the first place? No matter how complex the domain, the competence required to successfully perform tasks in that domain is represented in SLT in terms of finite sets of higher and lower order rules. These rules (defined in the next section) are called SLT rules to distinguish them from the more familiar production rules. The tasks themselves are represented in terms of input-output structures.

2. How do learners use and acquire knowledge? Why can some people solve problems for which they seemingly have all of the requisite knowledge, whereas others can not? Why do people often act in predictable ways even when they have many different options available? And, how is it that some people solve a problem almost immediately whereas others have to labor at the problem? Most would agree today that people have minds and in that sense construct their own knowledge. In SLT, people are assumed to solve problems and acquire knowledge by constructing (learning) new (SLT) rules, the latter (in oversimplified form) being accomplished by applying higher order rules to lower order rules.

3. How does one determine what an individual does and does not know? Behavior is the only thing that instructional scientists can observe. It is impossible to know all or exactly what any person does or does not know that causes him to behave as he does. Instructional scientists do not have license or the means to look inside. It is extremely unlikely, for example, that observing brain scans, even if legal, would ever provide the kinds of information necessary to predict how an individual might perform on any given task. Explaining complex human behavior requires higher level theory. In SLT, rules of competence are used as a standard for measuring individual behavior. Individual knowledge is operationally defined in terms of behavior judged relative to those competence rules.

4. How does knowledge change as learners interact with an external environment? How do initially naïve learners gradually come to acquire mastery? What are the fundamental

processes involved in longer term developmental changes? SLT attempts to answer these questions.

The remainder of this paper elaborates on the ideas and introduces new software technologies supporting the construction of instructional systems based on SLT.

STRUCTURAL LEARNING THEORY

Paralleling the initial four questions, SLT includes: a) a formal way to represent content as well as a formal method for constructing such representations, b) a cognitive theory, c) methods for operationally defining human knowledge in terms of behavior and d) a theory of instruction by which external agents (e.g., tutors) may influence learning.

THEORETICAL CONSTRUCTS

The following theoretical constructs are prerequisite to understanding SLT.

Observable Constructs. Specifying observable behavior might seem so obvious that it hardly needs elaboration. This may be true with computers where all inputs are reduced to ASCII characters, keystrokes and (e.g., mouse) events. It is not true in talking about human behavior. What serve as effective inputs and outputs depends inextricably on what can safely be assumed to act as inputs and/or outputs for the population of learners. (In the following exposition the term Ss is used for learners when referring to experiments.) An observable input or output (I-O) element in SLT can be anything, concept, process or even idea. Room, number, a spoken phrase, the SLT itself are all possible I-O. The primary requisite is that input or output must be atomic (directly perceived) elements.

Hierarchical Representation. Any I-O element can be represented as a hierarchy of elements at multiple levels of abstraction. That is, the same element can be represented as an atomic whole, or in terms of the components, categories or operations of which it is composed. Table 1 shows an I-O hierarchy of variables and their values at various levels of refinement. *Room*, for example, is more fully defined in terms of its components, *bed* and *carpeting*. *Carpeting*, in turn, is further decomposed into one or more scatter *rugs*.

TABLE 1

INPUT	
room <presentable, unpresentable>	*- component refinement*
bed <made, unmade>	*- atomic*
carpeting <clean,unclean>	*- prototype refinement*
rug <clean, dirty, messy&dirty>	*- atomic*
OUTPUT	
room <presentable>	*- component refinement*
bed <made>	*- atomic*
carpeting <clean>	*- prototype refinement*
rug <clean>	*- atomic*

In SLT I-O behavior can be described at any number of levels of abstraction. An essential constraint is that behavioral equivalence must be preserved at all levels of abstraction. In this example, *room* being *presentable* is precisely equivalent to *bed* being *made* and *carpeting* being *clean*.

Problem Domain. Problem domains in SLT are defined as sets of I-O pairs, including

outputs without specified inputs, that happen to be of interest to an outside observer (e.g., tutor). Problem domains may be well-defined or otherwise (i.e., ill-defined).

Well-defined I-O domains correspond to behavioral objectives (specifications in software engineering) and are semantically meaningful sets of inputs and outputs in which there is a unique output for each input. Simple examples include:

column subtraction problems --> differences
verbs --> participles (ing endings)
[real world object(s) + real world action] --> "subject" "verb"

Operations. Operations in SLT are processes, called SLT rules, which when carried out on given inputs in a domain generate unique outputs:

column subtraction
add "ing" and drop the final 'e' (as appropriate)
write the subject (name of object), then the verb with a "matching" ending.

Like I-O elements, operations (also called SLT rules) may be represented as hierarchies at multiple levels of abstraction. The above I-O hierarchy represents *room* at various levels of detail, both before and after some action (i.e., operation) has been performed thereon.

Such operations in SLT are called SLT rules -- to distinguish them from simpler condition-action (production) rules. SLT rules are represented as triples: Domain, Range (collectively an input-output Plan) and an Operation (or Procedure).

Figure 2 shows a hierarchy of behaviorally equivalent operations for cleaning rooms. There is an inverse relationship between the level of operations in this hierarchy and the complexity of the data structures associated with the corresponding parameters (in each operation's domain and range). For example, the *clean* operation operates on *room* as a whole, where *room* represents the full structure shown above. The lower level rule *make* operates on *bed* which in this example is a simple (i.e., atomic) element.

Problem: A problem consists of a given input structure, in which variables in the structure have been assigned values (e.g., *rug* is *dirty*), and the goal consisting of a data structure to which values are to be assigned (typically derived by applying operations). A Plan is a specification and is similar to a problem except that no input values are assigned.

Higher order rules play a central role in SLT. A higher order rule is a rule in which (some) inputs and/or outputs are themselves rules. One higher order rule, for example, might construct rules for solving different kinds of geometry construction problems from more basic operations (e.g., using a straight edge and compass, see Polya, 1960, Scandura, Durnin & Wulfeck, 1974). Another (in the mind of a great 19th century inventor) might convert (some domain of) manual operations (e.g., using brooms for cleaning) into an analogous one using electric power (e.g., vacuum cleaner).

Higher order problems and plans (specifications) also play a role in SLT (U.S. Patent, Scandura, 2000, in press).

CONTENT DOMAINS

The content associated with any given domain is defined in terms of observable behavior and the competence necessary to account for such behavior. As illustrated below, behavior is represented as hierarchies of inputs and outputs. Similarly, competence is represented

as hierarchies of behaviorally relevant rules.

Content is not restricted to well-defined domains. Ill-defined domains are arbitrary sets of I-O pairs, where the same input (in different contexts) may be paired with different outputs, where no inputs may be specified and/or where no one solution rule is known to exist. Many real world domains are of this sort:(a) chess positions and corresponding moves, (b) poems covering given topics and, (c) mathematical theorems and proofs.

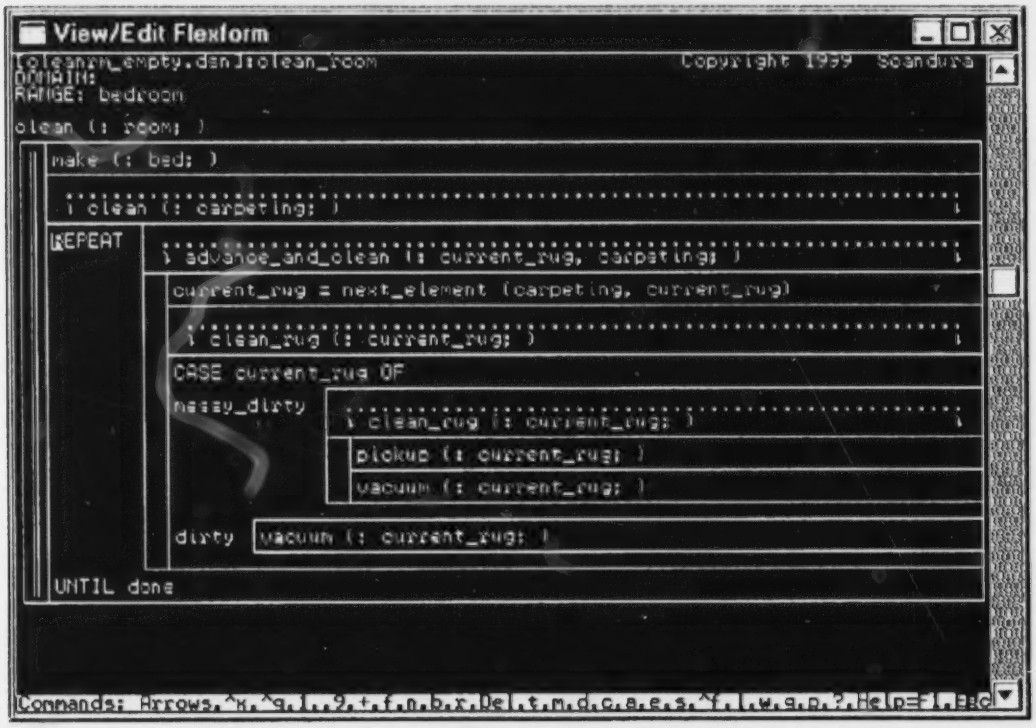

Figure 2. A sample hierarchy of SLT rules for cleaning rooms

To understand human behavior in such situations, it is essential to know what kinds of competence may be required. To better appreciate the approach taken in SLT, consider Artificial Intelligence (AI). In AI, the basic approach is to systematically search through a space of possible next (or previous) operations, trying and/or rejecting them, in turn, until a solution is found or the search space has been exhausted. The problem has been likened to constructing a bridge across a river on a foggy day, not knowing whether the river has made a turn. This strategy cannot succeed in its simplest form because the number of possibilities grows too quickly with the depth of the search space. While various heuristics may improve search efficiency, the methods used to identify such heuristics have been largely ad hoc.

SLT attacks the problem differently. Rather than asking how to combine a given set of components to find a solution, SLT asks, "What kinds of components are needed to solve problems (in the domain)?" Content analysis in SLT is based on a systematic process, called *Structural (cognitive Task) Analysis (SA)*. Unlike "knowledge engineering" in AI, which focuses on domain specific, SA is equally concerned with (e.g., higher order) rules that are domain independent. SA is necessarily carried out with some population of Ss and/or context in mind.

7 BEST COPY AVAILABLE

1. The first step in SA is to define the domain, in particular the observable inputs and outputs characterizing that domain. These inputs and outputs are represented as hierarchies. Identifying I-O is not always a trivial task. In teaching a non-swimmer to swim, for example, it is not immediately obvious that the inputs and outputs refer to operations. In this case, the task may be defined in terms of converting one (already known) operation -- namely floundering in the water-- to another (i.e., moving through the water while staying afloat by kicking and stroking with one's arms). Viewed in this way it is clear why starting the beginner with a "float" is often more efficient than starting without. There is less to learn! The learner never has to learn the equivalent of what is called the "dead-man's float" because the float serves to keep the person above water. Learning to kick and stroke serves the dual purpose of keeping the person afloat when the aid is later removed.

It is instructive to consider the role of (e.g., spatial) images in this context. The basic question is how to represent continuous reality in digital form. The first thing to observe is that images can also be represented hierarchically. Ignoring retinal imprints (and their associated physiology) images may be viewed as icons – symbols which share features in common with the reality they represent (e.g., Scandura, 1970). Such icons correspond to the top level in a hierarchy, representing say, a soccer field and the players on it. Lower levels in the hierarchy represent more detail – the players, position of the ball, etc.

Domains may even be specified entirely in terms of outputs. Consider situations where the inputs in question can only be determined in terms of the solution procedures used. Domains characterized by creative tasks, such as proving (or disproving) mathematical theorems, provide a case in point. Such proofs typically are based on large numbers of (input) assumptions and other theorems (lemmas, etc.), and often involve inordinately large search spaces. Other examples include chess where the same goal may be achieved from a variety of chess positions. Although outputs may be highly abstract in nature (e.g., writing an "interesting" poem), specifying outputs in a domain is a necessary minimum in SLT for predictive purposes. Otherwise, one is reduced to observing and reporting behavior after the fact.

2. The second step in SA depends on whether or not the problem domain is well-defined. If so, one or more solution rules (represented as hierarchies) may be constructed. If the domain is ill-defined, then one must first extract well-defined sub-domains of problems in the domain to serve as the starting point. Each well-defined sub-domain, in turn, will have at least one solution rule.

3. A hierarchy of rules is constructed for each well-defined domain. Highly systematic methods have been developed for this purpose[3]. Suffice it to say that each "slice" through a given competence hierarchy represents one of a set of behaviorally equivalent rules. Each rule in the hierarchy generates behavior equivalent to that associated with every other one.

The only difference is in the abstraction level of the operations involved and the complexity of the data structures associated with input and output parameters of the operations. Higher level rules operate on correspondingly more complex parameters. This inverse relationship is illustrated by the I-O data structures above (under Hierarchical Representation) and the clean room rule hierarchy shown in Figure 2. At the top of the hierarchy, for example, the *clean* operation in *clean (room)* is simple (and highly abstract) while the parameter *room* has a relatively complex structure. The next level operations, *make (bed)* and *clean (carpeting)* provide more detail as to process but correspondingly operate on less complex structures.

[3] A Patent detailing key aspects of the process was formally approved in November 2000 and is currently in press.

Rules for solving prototype problems in the well-defined sub-domains represent a first level characterization of the competence associated with the given problem domain. Starting with only the prototypes, however, typically leaves many "gaps" in complex domains. Many problems (I-O pairs) are generally not covered by any prototype.

4. SA offers an explicit way to fill those gaps. This is accomplished by first converting each rule into a higher order problem whose goal includes that rule. Higher order problems, in turn, are generalized to I-O specifications, or Plans, by ignoring the originally given values. Specifically, the higher order plans include previously identified rules in their ranges and/or domains.

5. A higher order rule is then constructed for solving problems specified by each higher order plan. In general, a higher order rule will not only generate the previously identified (lower order) rule, but other rules as well (depending on values assigned to input data elements). In effect, introducing higher order rules serves to fill gaps (in ill-defined domains). Furthermore, rules that can be generated from existing rules become redundant and may be eliminated from the set of rules characterizing competence (underlying the given domain).

This process of converting rules into higher order problems and constructing higher order solution rules may be continued indefinitely. At each stage, higher order rules become inputs and/or outputs for still higher order plans and rules. Empirical research (e.g., Scandura, 1984) demonstrates that the introduction of higher order rules, and the elimination of redundant ones, has two beneficial effects: a) Individual rules (including higher order ones) tend to become simpler and b) the collective generating power of the rules becomes greater. That is, greater varieties of problems in the original domains can be accounted for via the identified rules. Repeating the process makes it possible to account for arbitrarily and sufficiently broad ranges of tasks in the problem domain with sufficiently small sets of higher and lower rules.

Discussion. There are indefinitely large numbers of real world domains. Analyzing such domains would correspondingly require an indefinitely large amount of work -- especially if one had to start SA from the beginning with the introduction of each new domain. In fact, this is *not* necessary. SA is a strictly cumulative process. Many rules, particularly higher order rules, are applicable across multiple domains. Rather than having to start over with each new kind of problem, one can build on the results of previous SA: Simply add the new problems to the old domain, introducing new prototypes as necessary. The process of SA continues as before.

To summarize, competence underlying given I-O domains is represented by a set of higher and lower order rules, where each rule is represented as a hierarchy. Each hierarchy represents an equivalence class of rules, each representing a different level of automation. Given any domain characterizing the behavior of interest, SA offers a systematic means of constructing competence for solving "sufficiently broad" sets of problems in the domain.

The same domain may be analyzed from any number of perspectives. That is, any problem in a given domain may be solvable in any number of different ways (using rules associated with the differing perspectives). The fact that European students are commonly taught to solve column subtraction problems by equal additions while American students use borrowing is well documented in the literature (e.g., Durnin & Scandura, 1973). In practice, the number of different perspectives required for instructional purposes is usually quite small. In most domains, small numbers of alternative rule sets are sufficient to account for the behavior of most learners.

In performing SA, it also is important to recognize that what serve as inputs and outputs may vary across apparently similar sub-populations of learners. Different sub-populations, for

example, may make differential use of spatial versus non-spatial inputs in direction finding.[4] Similarly in soccer, for example, a valid representation will depend on whether the field is viewed from the perspective of a coach (with all players viewed from a common perspective) or one of the players. In SLT, specifying what are the effective inputs and outputs often goes a long way in defining what is to be learned.

SA has been used in practice for many years, and numerous examples exist in the literature (Scandura, 1982; Scandura, 1977c; Scandura & Scandura, 1980; Scandura, 1997; Scandura & Scandura, 1999). These include both broad-based comprehensive analyses (Scandura, Durnin, Ehrenpries et al, 1971; Scandura & Scandura, 1980) and smaller more intensive analyses (e.g., Scandura, Durnin & Wulfeck, 1974; Wulfeck & Scandura, 1977). SA has evolved considerably since its inceptions in the 1960s through the early 1980s (e.g., Scandura, 1964a,b; Scandura, Durnin & Wulfeck, 1974; Scandura, 1977c; Scandura & Scandura, 1980; Scandura, 1984b). Indeed, the method itself had become increasingly systematic and repeatable over the years, leading today to precise automated methods in software engineering (U.S. Patent Scandura, 2000, in press). Other early references of interest include (Roughead & Scandura, 1968; Scandura, 1968a,b, 1969a, 1969b; Scandura, 1971b; Scandura, 1972).

COGNITION

The author's epistemological assumptions are constructivist in nature. Knowledge acquisition is an active process. People are assumed to have minds; they learn and otherwise construct new knowledge. Unlike most constructivist theories, however, SLT is deterministic in character and firmly rooted in behavior (Scandura, 1971b, 1973, 1977a,b,c, 1978a,b). In the SLT, cognition has always been characterized in terms of lower and higher order rules operating under universal constraints. Lower and higher order rules correspond to domain-specific and general-purpose knowledge, respectively. On the other hand, all rules have the same form. The only difference between lower and higher order rules is that the latter contain other rules in their domain or range. Specific rules of knowledge correspond to one [or more] specific levels in a rule hierarchy.

The use of and interactions between known rules are governed by a universal control mechanism, and constrained by an assumed fixed processing capacity and a characteristic processing speed. In its original form (e.g., Scandura, 1971b; Scandura, 1973) control was characterized in terms of goal switching: When faced with a problem, the learner is assumed to search the set of rules in his processor (often called short term memory) to see if one applies. If so, the rule is applied.[5]

If not, control is assumed to move to the higher level goal of deriving a rule that does apply. The higher order rule is then applied, generating a new rule, with control reverting to the next lower level. To wit, faced with a problem for which no solution process is known, attention focuses on constructing one. The process is assumed to continue until an appropriate rule is found.

This control mechanism was shown to work both when no rule applies as well as when more

---

[4] Such differences may correlate with gender (due to well-documented hemispheric differences in male and female brains). Although such differences would be of interest in defining sub-domains of problems, SLT puts the emphasis on individual behavior as opposed to group averages.

[5] Concept attainment, for example, provides one, perhaps less than obvious application of this idea. Bruner, Goodnow and Austin (1956) showed how concepts could systematically be attained by applying a uniform strategy to a given a set of instances (examples) of the concept. Scandura (1964a) extended this idea to more complex kinds of concept learning based on abstract card tasks.

than one rule applies (Chapter 8 in Scandura, 1973). Later, it also was shown to account for automatization processes whereby more efficient rules are derived from known rules (Scandura & Scandura, 1980). Supporting experiments were conducted under a wide range of conditions (e.g., Scandura, 1971a; Scandura, 1973, Scandura, 1974; Scandura, 1977c; Scandura & Scandura, 1980). Many of the experiments were run under carefully controlled conditions, where it was clear exactly what Ss knew on entry into the experiments and where memory was explicitly eliminated as a factor.

Although suggestive, goal switching is extremely difficult to formalize in a way that is completely independent of the higher order rules involved. Wulfeck and Scandura (Scandura, 1977, Chapter 14), for example, were able to successfully simulate the growth of problem solving behavior over time. Goal switching control and core higher order rules, however, were subtly but inextricably intertwined in the simulation (program).

The search for a way to formally represent goal switching, with control completely separate from higher order rules, has been long and hard. Rather than goal switching as such, it gradually became clear that each level of "goal switching" might better be characterized in terms of searching the (input) domains and ranges of (higher order) rules for (lower order) rules that match given problems at different levels of embedding. For example, if no solution rule is found for solving (i.e., containing) the given problem, then the search continues for rules whose outputs contain such rules. In the case of failure, the search moves to deeper levels. Each search level seeks rules whose ranges in which needed solution rules are embedded (at some level). A recent patent (Scandura, 2000, in press) also shows how searching for plans (rules without operations) and higher order plans also plays an important role in accounting for human behavior -- the commonly observed human behavior of formulating (new) plans before their implementation.

Other studies offer support for the assumption that each individual has a fixed processing capacity (e.g., number of rules that can be kept in mind at one time) -- irrespective of the task in question (Voorhies & Scandura, Chapter 7 in Scandura, 1977). This result represents an extension of Miller's (1956) classic result indicating that people on average can process between seven plus or minus two chunks of information. Support for a fixed processing speed is more hypothetical and an area in which definitive research is needed. Nonetheless, everyday observation suggests that individuals differ in the rate at which they process information. Some people characteristically talk faster than others, for example, while others are more characteristically deliberate. Indeed, such information is commonly used, often informally, in assigning people for various tasks. Research is needed to determine how much of such behavior is due to innate characteristics and how much to what has been learned. One hypothesis, for example, is that faster innate processing leads to greater numbers of more specific rules whereas characteristically more deliberate behavior leads to more highly structured general-purpose higher order rules.

To summarize, the kind of behavior associated with any given problem necessarily depends on what the learner knows at the time. Problem solving is necessary when no solution rule is immediately available. Higher order rules are assumed to play a key role in deriving solution rules. Behavior is simpler when the learner already knows a solution rule. Learning does not stop there, however. Other higher order rules are used for automatization. Procedural complexity in rules is successively replaced with structural complexity as behaviorally equivalent rules in a competence hierarchy are constructed (learned). The more complex the domain and range structures of a solution rule's parameters, the more rapid and more flexible the behavior. Automated rules correspond to rules at (or near) the top of the competence hierarchies to which they belong (see Content).

Top level rules (in competence hierarchies) are atomic and, effectively, are operational versions of what is commonly called *declarative knowledge*. Interactions between higher

order automatization rules and rules lower in a competence hierarchy are governed by the same "goal switching" control. The result of such interactions are higher *level* rules -- not higher *order* rules. In the section on transitions, automatization is shown to close a theoretical loop of sorts, making it possible to explain, predict and even control learning over long periods of time. Studying the latter has been largely the exclusive domain of developmental psychologists but it is equally important to instructional scientists.

Higher order rules also play a crucial role in rule selection (e.g., in studies of "interest", "direction" or "motivation") where more than one solution rule is available. In particular, the control mechanism directs the search to higher order rules, irrespective of whether no matching rule is found, or whether two or more rules are found. In the latter case, the subject is forced to choose. Rather than introducing a fixed (and usually arbitrary) conflict resolution technique (as is commonly done in expert systems), resolution in SLT is handled in the same manner as all problem solving -- via higher order selection rules. Irrespective of the selection rules introduced, the same control mechanism is used. This fact was first noted in the literature in (Chapter 8 in Scandura, 1973; compare with Scandura, 1971b).

Higher order automatization rules are also governed by the same principles, and have been shown to account for the kinds of behavior associated with Piaget's developmental stages (Scandura & Scandura, 1980). Analysis in terms of rules and higher order automatization rules provided an efficient and precise account for transitions from pre-operational to concrete operation stages of development. Until pre-operational rules had been mastered, it was impossible for children to learn what Piaget had defined as concrete operational behavior. Furthermore, the analysis also accounted for the problem of horizontal decalage. Pre-operational behavior was observed where and only where the requisite pre-operational behaviors had been mastered.

The latter problem had long defied easy explanation within both Piagetian and behavioral frameworks. Automatization leads to the acquisition of rules at progressively higher levels in hierarchies of behaviorally equivalent rules. Ultimately, the top rule is learned. Elementary operations (procedural knowledge) associated with top level rules are mathematically equivalent to simple relations (declarative knowledge). What was originally procedural knowledge becomes declarative in nature (i.e., capable of serving as a basis for new learning. This is true not only in a formal (i.e., mathematical) sense but also behaviorally. Once automated, declarative knowledge effectively defines new I-O elements. These new elements provide the grist for entirely new kinds of learning (as with Piaget's stages -- e.g., see Scandura & Scandura, 1980, p. 127). This point is further elaborated below in the section on transitions.

ASSESSING BEHAVIOR POTENTIAL

Given what a learner knows and his current state of knowledge, the above analysis provides a basis for predicting that learner's behavior on any given problem. However, this is rarely known; and one certainly cannot know everything. One cannot look inside a person's head to determine what is known, as one might a computer program (inside a computer). We can only infer what is there by observing the learner's behavior.

The basic question is how to most efficiently determine what a given learner knows at any given point in time. Three basic problems from a SLT perspective are known.

1. How to distinguish alternative accounts of the same behavior (e.g., determine the "best fit" between borrowing and equal additions in column subtraction -- see Durnin & Scandura, 1973)?

2. How to distinguish known and unknown paths in rules in a given rule hierarchy (e.g., the ability to subtract but not when there are zeros in the top numeral -- see Scandura, 1971b; Scandura & Durnin, 1978; Durnin & Scandura, 1973; Hilke, Kempf & Scandura, 1977; Scandura & Reulecke, Chapter 10 in Scandura, 1977)?

3. How to distinguish the atomicity level of equivalent rules (slices) in a given hierarchy? The problem is how best to distinguish between behaviorally equivalent rules, in which the only difference is in process atomicity and corresponding structural complexity (with top level or atomic rules corresponding to direct declarative knowledge).

Distinguishing between learned paths and levels in a given abstraction hierarchy is relatively well understood and researched (e.g., Durnin & Scandura, 1973). Particularly relevant to instructional design is making knowledge assessment more efficient by testing at multiple levels in a hierarchy (e.g., Scandura, Stone & Scandura, 1986; Scandura, 1987b; Scandura & Scandura, 1987). This approach uses the principle of "splitting the difference" -- selecting those test items at each step that provide the most information about which paths are learned and which are not. For example, if a person is successful on a path in the middle of a hierarchy, then he is almost certainly able to solve tasks associated with "easier" paths (in which learned components are refined into more elementary components). Conversely, failure would imply failure on all paths that are more difficult.

Although the problem of distinguishing alternative accounts of the same behavior has been addressed in earlier research (e.g., Durnin & Scandura, 1973; Reuleuke & Scandura Chapter 10 in Scandura, 1977), more explicit theory and basic experimental research is needed. Similarly, although response time has been widely used in psychological experiments to distinguish alternative accounts of behavior, this research might well be put on a firmer theoretical basis if higher order processes responsible for atomicity were given more explicit consideration (e.g., Scandura & Scandura, 1980).

Assessment problems become even more complex when seeking to determine what is learned with respect to sets of rules collectively. The number of possible distinctions becomes much greater and assessment considerably more difficult. A key question is whether one should bother with such situations at all. Theoretically, it should be possible to obtain the same information by assessing higher and lower order rules individually. Implementing this approach, of course, requires constructing suitable test items for given higher order rules. The feasibility of doing so has been demonstrated in laboratory experiments (e.g., Scandura, Durnin, Ehrenpreis et al, 1971; Scandura, 1974).

TRANSITIONS

The above analysis (under COGNITION) provides a basis for refining answers to basic questions, "Why is it that some people can solve problems that others cannot? And, how is it that initially naïve learners acquire new competence -- and gradually come to acquire the mastery associated with experts?" (Scandura, 1981, p. 139). The SLT control mechanism, together with requisite rules, was shown to provide a uniform basis for explaining transitions from naïve problem solver to neophyte. The same mechanism, together with higher order automatization rules, was also shown to account for mastery. Once one rule in a competence hierarchy has been learned, automatization results in the acquisition of higher level rules – ultimately leading to top-level atomic operations with fully elaborated data structures.

For illustrative purposes, consider the task of constructing a statement expressing one's limited ability to speak German. The basic problem (given → goal) can be represented as:

[Idea: Know little German] → <?Proper descriptive phrase>

The naïve learner approaches the problem with a knowledge base consisting of rules for expressing ideas associated with terms like "ich" [I], "Deutsch" [German], "ein wenig" [a little], "sprechen" [to speak], "kann" [can], "bin" [am] and "nur" [only]. In order to construct an appropriate response, the learner would also need some equivalent of a higher order rule to the effect, "put words in the order: subject, initial verb, adjectives and objects, other verbs".

The neophyte's knowledge base might directly include a statement to the effect "Ich kann nur ein wenig Deutsch sprechen" [I can only a little German speak.] More precisely, this knowledge is represented by a rule for constructing such a statement.

It is almost an oxymoron to refer to a master's knowledge base in this context, so we assume the master is a German teacher. The master's knowledge base is likely to include a wide variety of phrases with very similar meanings: "Ich kann nur ein wenig Deutsch sprechen", "Ich bin im Deutschen ein Anfanger" [I am in German a beginner], and so on. The master's data structures are far more complex, representing a variety of (sub-categories of) meanings and corresponding expressions, enabling the expert to express fine distinctions unavailable to the neophyte.

More generally, transitions from naïve to neophyte status are characterized by the use of higher order rules to construct new rules. Once learned, a new rule allows the neophyte to solve similar problems in a systematic manner. Transitions from neophyte to master status involve application of higher order automatization rules. These higher order rules transition learning from one level in a rule hierarchy to another. They generate higher level (behaviorally equivalent) rules with simpler operations and correspondingly more complex parameters (data structures).

Application of SLT to Piagetian research (Scandura & Scandura, 1980) exposed more global transitions where mastery in one domain made it possible to learn qualitatively different domains. What constitute rules to be learned in one domain effectively become -- once mastered -- input and output elements for defining entirely new domains. Mastered rules in one domain provide elements for the next.

For example, it is often a painstaking process for young children to learn to construct Arabic numerals, 1, 2, 3, 4, 5, ... by combining previously learned straight and curved line segments in precise patterns. Gradually, however, the process becomes increasingly easier, faster and more reliable. Ultimately, writing numerals becomes fully automatic, taking only the time required by the physical actions themselves.

Once mastery has been achieved, these former operations effectively become declarative (elemental in nature). Writing numerals can now serve as input and output elements for defining basically new kinds of problems. Teaching a young child how to perform column subtraction, for example, would be meaningless unless the child first knew how to write numerals (or perhaps to type with the aid of a computer). Similarly, it is hard to imagine a person learning to write poems, for example, without having first mastered a reasonable vocabulary

Scandura and Scandura (1980) found that transition from Piaget's pre-operational stage of development to his stage of concrete operations was possible only after certain operations had been fully mastered (i.e., automated). True number conservation, for example, was impossible until Ss had mastered one-to-one mappings. Until then, number conservation problems were effectively meaningless. According to SLT these principles are universally

applicable, and play an essential role in all such transitions. Mastering one domain provides a new foundation on which subsequent learning may be based.

TEACHING AND LEARNING

This section returns briefly to Figure 1, and particularly to what is meant by "diagnostic and tutorial expertise". In oversimplified form, tutorial expertise means asking the right questions and presenting the right information at the right time relative to what the learner knows (Scandura, 1964b; Scandura, 1987a; Scandura & Scandura, 1987). Methods used to teach various kinds of content derive from a common set of SLT principles. Optimal instructional techniques depend as much on the learner's state (of knowledge) as on the kind of content (e.g., conceptual, procedural, problem solving).

Assuming input and output prerequisites, and well-defined competence, for example, optimized assessment involves choosing test items that maximize the amount of information that can be gained about the learner's state (of knowledge). Initially, the tutor has no idea of what the learner does and does not know. All paths in all rules in the target competence hierarchy are question marks. At each stage of learning, the tutor should choose that test item providing the most information. Initially, this item will be associated with some path in a rule located midway in the hierarchy. Success here implies success on all subsumed paths of rules lower in the hierarchy. Conversely, failure implies failure for all super-ordinate (encompassing) paths in rules higher in the hierarchy. Irrespective of success or failure, maximum information is attained about what the learner knows.

Maximal information also is attained about what the learner needs to know in order to overcome any inadequacies. Instruction should optimally be associated with a path in the hierarchy for which all prerequisite components have been learned. The process of "splitting the difference" in testing (to obtain maximum new information) and providing just that information needed (to build on what is already known) continues until all paths have been learned. This optimized process for testing well-defined (procedural) content, herein referred to as a General Purpose Tutor(GPT), is represented as a Flexform in Figure 3 (Scandura, Stone & Scandura, 1986; Scandura, 1987a; Scandura & Scandura, 1987).

BEST COPY AVAILABLE

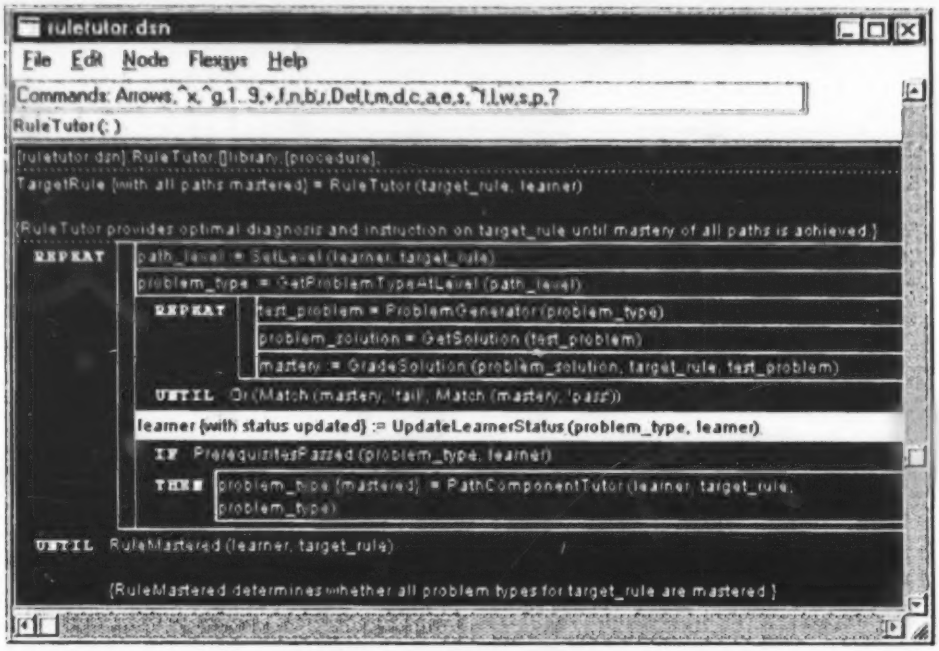
ruletutor.dsn
File Edit Node Flexsys Help
Commands: Arrows, ^x,^g,1..9,+,f,n,b,s,Del,t,m,d,c,a,e,s,^l,l,w,s,p,?
RuleTutor(;)
[ruletutor.dsn].RuleTutor.[]library.[procedure].
TargetRule {with all paths mastered} = RuleTutor (target_rule, learner)
{RuleTutor provides optimal diagnosis and instruction on target_rule until mastery of all paths is achieved.}
REPEAT
path_level = SetLevel (learner, target_rule)
problem_type = GetProblemTypeAtLevel (path_level)
REPEAT
test_problem = ProblemGenerator (problem_type)
problem_solution = GetSolution (test_problem)
mastery = GradeSolution (problem_solution, target_rule, test_problem)
UNTIL Or (Match (mastery, 'fail'), Match (mastery, 'pass'))
learner {with status updated} := UpdateLearnerStatus (problem_type, learner)
IF PrerequisitesPassed (problem_type, learner)
THEN problem_type {mastered} = PathComponentTutor (learner, target_rule, problem_type)
UNTIL RuleMastered (learner, target_rule)
{RuleMastered determines whether all problem types for target_rule are mastered.}

Figure 3. The General Purpose RuleTutor represented as a Flexform in Softbuilder.

This GPT is entirely independent of the particular rule to be taught. Given a suitable representation of any rule, the tutor can select those test items at each stage of the process, which provide optimal amounts of information about the learner's state. It can also identify exactly the information that the learner needs most. Perfect memory and rapid processing speeds would enable an automated tutor to do this more efficiently, and with more reliability than the best human teacher. The information that must be referenced with complex content at each stage of the process is likely to go far beyond what any human might reasonably be expected to keep in mind.

These same principles may be applied in teaching problem solving skills. Higher order rules may be taught in the same manner as other rules. Although discovery learning is often an assumed prerequisite to higher order knowledge, research shows that what is learned depends far more on what is taught than on the way it is taught (Scandura, 1964a; Scandura, 1964b). The main determinate is the degree to which the target content has been identified and represented in operational terms (as rules). Higher order rules, for example, can be taught by exposition as efficiently as by discovery (Roughead & Scandura, 1968). The only caveat is that the higher order rules in question must be specified with sufficient precision to enable (direct) instruction. To date, little attention has been given to isolating requisite higher order rules.

What seems to matter most is what a learner knows when information is presented. If given to soon, information goes unused, and can even be misleading. Given too late and the learner can become discouraged, disoriented and/or otherwise loose interest (Scandura, 1964b, Scandura et al, 1969b). Although higher order rules have rarely been taught directly in problem solving instruction, empirical research suggests to the extent the higher rules can be precisely identified that this would be a very efficient way to proceed.

BEST COPY AVAILABLE

Problem solving behavior is commonly taught by example plus practice.[6] Optimized instructional decision making becomes considerably more complex in this context. Learners typically use various combinations of higher and lower order rules, and it is very difficult both to choose appropriate test problems and to determine the most helpful information.

The GPT design is extended in Figure 4 to accommodate arbitrarily complex (e.g., problem solving) domains. It is designed to accommodate the learner's use of various combinations of higher and lower order rules in attacking test problems. Assessing higher order knowledge in this context depends on pre-specification of requisite higher order rules.

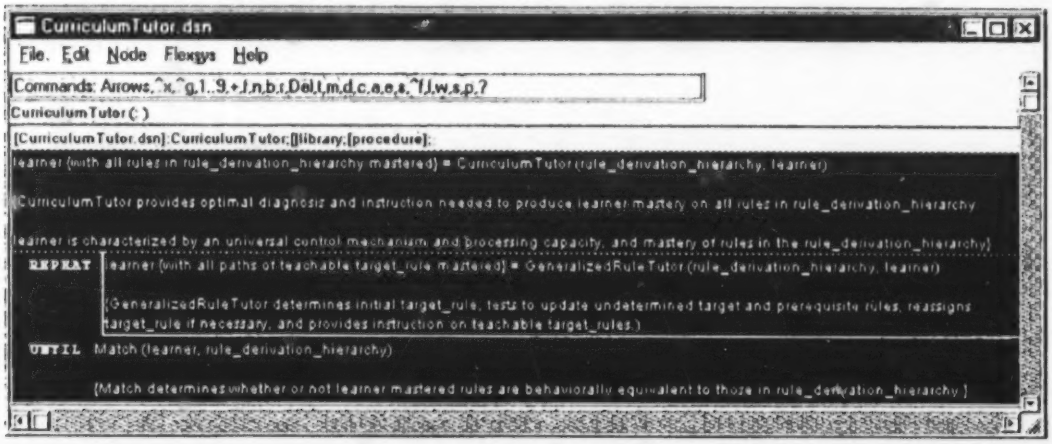
CurriculumTutor.dsn
File Edit Node Flexys Help
Commands: Arrows,^x,^g,1..9,+,/,n,b,r,Del,t,m,d,c,a,e,s,^f,l,w,s,p,?
CurriculumTutor( )
[CurriculumTutor.dsn]:CurriculumTutor;[]library;[procedure]:
learner (with all rules in rule_derivation_hierarchy mastered) = CurriculumTutor (rule_derivation_hierarchy, learner)
{CurriculumTutor provides optimal diagnosis and instruction needed to produce learner mastery on all rules in rule_derivation_hierarchy
learner is characterized by an universal control mechanism and processing capacity, and mastery of rules in the rule_derivation_hierarchy}
REPEAT learner (with all paths of teachable target_rule mastered) = GeneralizedRuleTutor (rule_derivation_hierarchy, learner)
{GeneralizedRuleTutor determines initial target_rule, tests to update undetermined target and prerequisite rules, reassigns target_rule if necessary, and provides instruction on teachable target_rules.}
UNTIL Match (learner, rule_derivation_hierarchy)
{Match determines whether or not learner mastered rules are behaviorally equivalent to those in rule_derivation_hierarchy.}

Figure 4. The General Purpose CurriculumTutor represented as a Flexform in Softbuilder.

As shown above (CONTENT), Structural Analysis (SA) can be applied to ill-defined as well as well-defined content domains. In principle, SA can also be applied dynamically as unanticipated issues arise during the course of instruction. A learner might, for example, ask or answer a question that adds a new dimension to any given SA. The tutor has the option of extending the domain of discourse by dynamically extending the content domain and SA thereon. The ability to dynamically extent the domain of discourse during instruction is something that separates superior teachers from others. It is not clear, however, to what extent such abilities can be automated.

The time-honored way to assure that a rule is mastered is practice. Research (as well as common observation) shows that repeating a learned solution results in faster surer responses. The current analysis suggests that more might be done by way of pinpointing higher order automatization rules, and teaching them to help learner's automate previously learned rules. Although supporting research is limited (Scandura & Scandura, 1980), SLT suggests that automatization can be made more efficient by identifying and explicitly teaching automatization techniques. The idea is to identify higher order rules that assist learners to transition from rules lower in a set of competence hierarchies to rules at higher levels. (Note: The distinction between *higher order* and *higher level* is crucial. Any given higher order automatization rule may effect transitions to higher levels in an indefinitely large number of rule hierarchies.)

[6] Many mathematicians believe that the only way to become a good problem solver is to solve a lot of problems.

BEST COPY AVAILABLE

TECHNOLOGY

Implementing the kinds of tutorial systems described above would be relatively difficult with currently available technologies. Although major advances have been made in recent years in component based software and object oriented programming, for example, current methods make them too complex and difficult to use by most instructional scientists. Indeed, many working in theoretical computer science tend to avoid programming, precisely because of its idiosyncrasies and complexity. Things that are not easily learned without spending inordinate amounts of time and effort.

Scandura.com's Softbuilder system approaches this problem with its abstract syntax tree (AST) based high level design (HLD) language. ASTs are widely used in artificial intelligence because they directly reflect real world data structures and operations (e.g., Scandura, 1987b, 1991,1992, 1994). The need for and use of such programming constructs as data types, global variables, semi-private methods and the like is eliminated entirely. Softbuilder also ensures component interoperability and extensibility (i.e., the ability to add new components as they are needed).

Softbuilder's ability to model the real world on its own terms makes it possible for computer literate instructional scientists (who are not also programmers) to directly participate in building automated tutorial systems.

AutoBuilder research goes further in this direction. In addition to enabling domain experts (as well as programmers) to more efficiently build and maintain complex software systems, AutoBuilder will guarantee that the software will operate exactly as specified. AutoBuilder is based on recently patented processes (Scandura, 2000, in press) and a prototype in which software specifications and designs are represented in semantically meaningful terms at multiple levels of abstraction. These abstraction hierarchies are guaranteed to be internally consistent, with designs guaranteed to be correct with respect to specifications.

A software system built with AutoBuilder is guaranteed to work as a whole if each component in which it is implemented performs as specified. Implementation components can be made arbitrarily small, thereby making it easy (either by inspection or automated theorem provers) to ensure that they are correct. A sample AutoBuilder refinement is shown in Figure 5. (Figure 2 above shows an internally consistent design hierarchy for cleaning rooms.)

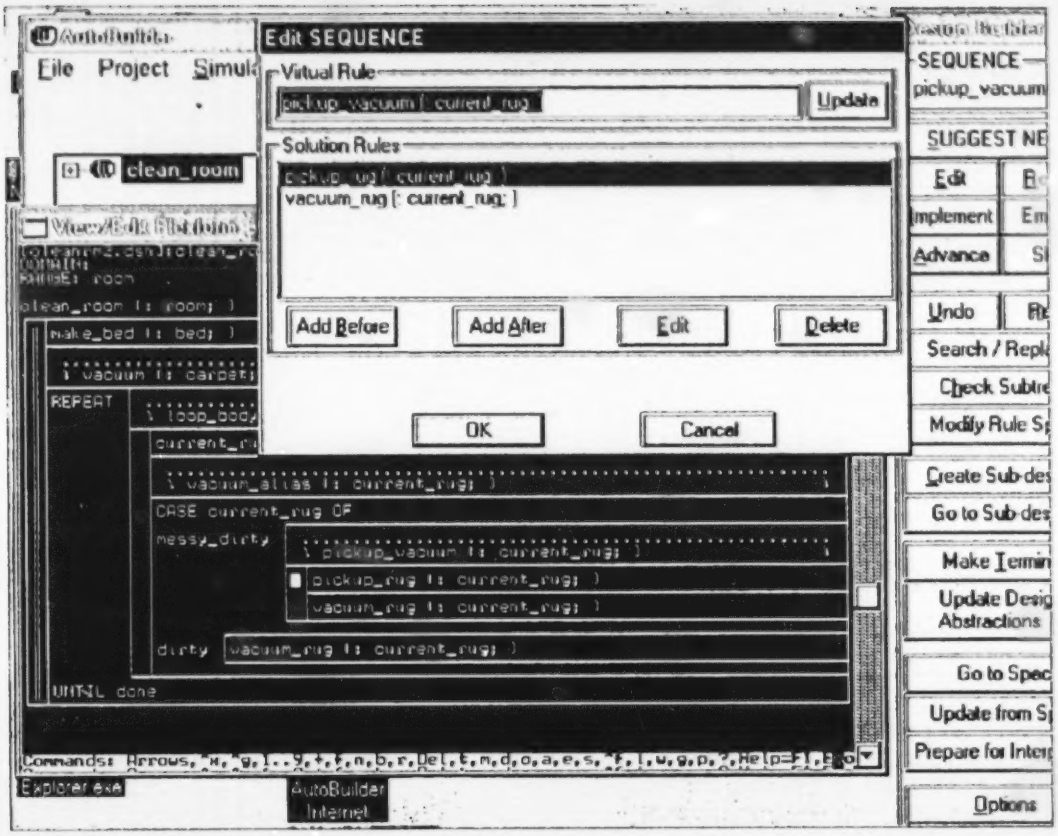

Figure 5. AutoBuilder in the process of refining the virtual component, pickup_vacuum (:current_rug;), into a sequence.

Softbuilder and AutoBuilder will play an essential role both in representing the content to be taught and in constructing general-purpose tutors (see Figure 6). Softbuilder includes specific support for interacting systems -- for example, interacting tutors and learners. At the highest level of abstraction an Intelligent Tutor may be represented as a dialog between a tutor and a learner with respect to given content.

dialog (content, tutor, learner)

The tutor and learner are independent agents, which communicate with one another via the content. At the top level of the abstraction (in a hierarchy) these "agents" may be characterized as follows:

learner (tutor_event:; learner_event)

learner is a fully independent agent reacting to *tutor* created events but independent of *tutor* per se.

tutor (learner_event: learner_model; tutor_event)

BEST COPY AVAILABLE

An automated *tutor* must know what *learner* knows at each stage of learning in order to know what question to ask and how to update his model of *learner* based on *learner*'s answers.[7] As discussed above, *content* may be well-defined, ill-defined or dynamic.

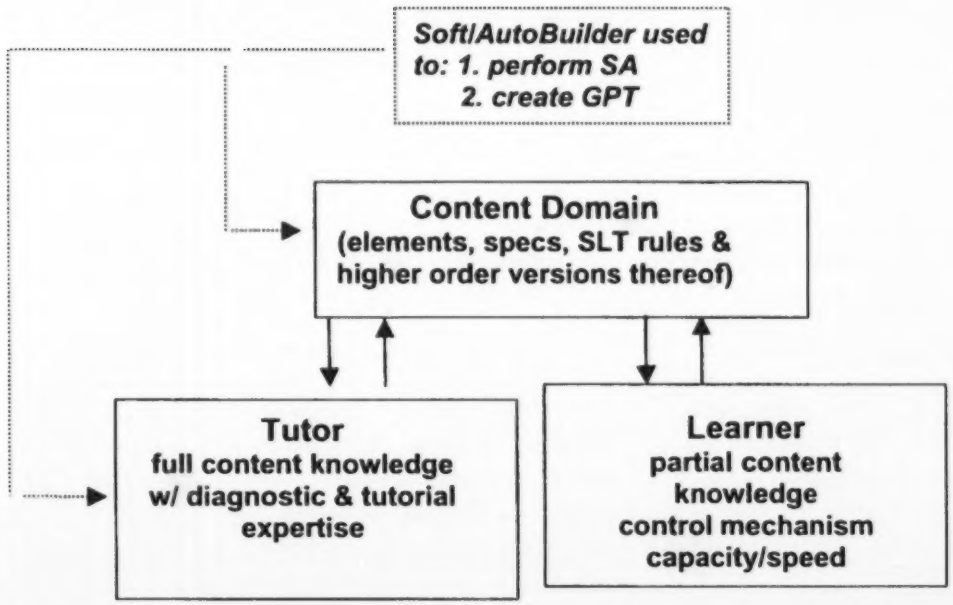
Soft/AutoBuilder used to: 1. perform SA 2. create GPT
Content Domain
(elements, specs, SLT rules & higher order versions thereof)
Tutor
full content knowledge w/ diagnostic & tutorial expertise
Learner
partial content knowledge
control mechanism
capacity/speed

Figure 6. Roles Softbuilder and AutoBuilder will play in construction of the general-purpose tutors described above.

[7] The aforementioned analysis of tutor-learner interactions can be generalized to any set of interacting intelligent agents. Interactions between interacting groups of individuals might be formulated using the same basic machinery. For example, the soccer players in recent research by Furbach, Stolzenburg, Baumgartner, Obst et al[7] interact at multiple levels and, hence, might be conceptualized in this manner. See Appendix B for more details and related analysis.

CONCLUDING REMARKS

This paper updates and extends the early SLT literature, with emphasis on providing a more complete and coherent picture of SLT. Many instructional designers, for example, have read only the limited version published in Reigeluth (1983). Although many of the ideas expressed herein are natural extensions of the earlier literature, much of that literature is scattered and otherwise inaccessible. Similarly, while motivation for many of the ideas may be found in the early literature (e.g., Scandura, 1971b, 1973, 1977c; Scandura & Scandura, 1980), other key ideas have been developed only recently (e.g., Scandura, 1997). Higher order rules have played a central role in SLT since its inception. This paper further shows how SLT offers a unified theory of cognition, making it possible in principle to explain all behavior from birth onward. Higher order knowledge produces new knowledge, which in turn gradually becomes more automated. Automatization, in turn, introduces new I-O elements, allowing the process to repeat at higher levels of learning.

A recent 300 plus page patent (Scandura, 2000, in press) details SA and the universal control mechanism. There are many areas, of course, where further refinement and extension is needed.

Although earlier research will be suggestive in many cases, more work is particularly needed in knowledge assessment. Distinguishing between learned paths and levels in given abstraction hierarchies seem to be well-understood and researched, although many of the ideas should be used more widely in practice. Making knowledge assessment qualitatively more efficient via integrated testing at multiple levels in a hierarchy seems ripe for broad applications. More explicit theory and basic experimental research is needed in other areas. For example, more explicit methods are needed for distinguishing alternative accounts of the same behavior as well as for distinguishing behaviorally equivalent SLT rules in a hierarchy.

In its current state, SLT can and should be applied far more widely in instructional design. Indeed, major parts of structural analysis have been and are being applied successfully in software engineering, and it is possible to envisage broad areas of SLT application in artificial intelligence and intelligent systems generally. SLT could even motivate formal developments in these areas[8].

In summary, this paper covers a lot of ground: a) how previously acquired and/or innate knowledge (including elements and rules) are prerequisite to new learning, b) how that new knowledge is constructed (by interactions between lower and higher order rules), c) how mastery (via higher order automatization rules) transitions procedural into declarative knowledge and d) how the process repeats at higher developmental levels, with declarative knowledge forming new elements on which still higher level tasks may be defined. This process has the potential of accounting for complex learning from birth onward.

The paper also shows that what a learner does and does not know can be determined by observing behavior, and how the information gained provides a foundation for highly efficient teaching and learning. The paper also touches on the kinds of technology necessary for implementing such theory in constructing automated instructional systems and outlines relationships to software research.

Nonetheless, much was omitted due both to space and time limitations. More explicit reference to the previous literature would have been helpful to serious scholars in the area, and certainly more attention could have been given to examples.[9] Many examples have been detailed in the earlier literature but some would clearly benefit from reformulation in

---

[8] See Baumgartner, P. Theory Reasoning in Connection Calculi. Berlin: Springer, 1998

[9] Appendix A provides a very useful roadmap to the earlier literature. Also see Scandura (1996).

modern terms. The current SLT formulation, for example, is more uniformly couched in terms of abstract syntax trees, which are used to represent both data elements and operations (and in turn rules, plans and problems).[10]

Despite these limitations, I hope the reader will come away with some sense of the exciting new possibilities SLT offers for fundamental research in both cognition and instruction -- and more important that some readers will be motivated to apply and/or extend SLT in the future.

[10] Appendix B suggests how SLT motivates more formal developments in computer science, but had to be omitted to conserve space. It is included at www.scandura.com under Knowledge Base.

References

Bruner, J.S., Goodnow, J.J. and Austin, G.A. A Study of Thinking. NY: Wiley, 1956.

[63] Dumin, J.H. & Scandura, J.M. An algorithmic approach to assessing behavior potential: Comparison with item forms and hierarchical analysis. Journal of Educational Psychology, 1973, 65, 262-272.

[94]Hilke, R., Kempf, W.F., & Scandura, J.M. Deterministic and probabilistic theorizing in structural learning. In H. Spada & W.F. Kempf (Eds.), Structural models of thinking and learning. Bern: Huber, 1977.

Miller, G.A The magic number seven, plus or minus two. Psychological Review, 1956, 63, 81-97

Polya, G. Mathematical Discovery. New York, 1962.

[38]Roughead, W.G. & Scandura, J.M. "What is learned" in mathematical discovery. Journal of Educational Psychology, 1968, 59, 283-298.

[2] Scandura, J. M. Abstract card tasks for use in problem solving research. Journal of Experimental Education, 1964, 33, 145-148. (a)

[3] Scandura, J. M. An analysis of exposition and discovery modes of problem solving instruction. Journal of Experimental Education, 1964, 33, 149-159. (b)

[37]Scandura, J.M. Research in psycho-mathematics. The Mathematics Teacher, 1968, 61, 581-591. (Reprinted in The Mathematics Education, 1968, 3. (a)

[43]Scandura, J.M. New directions for theory and research on rule learning: I. A set-function language. Acta Psychologica, 1968, 28, 301-321. (b)

[44]Scandura, J.M. New directions for theory and research on rule learning: II. Empirical Research. Acta Psychologica, 1969, 29, 101-133. (a)

[45]Scandura, J.M. New directions for theory and research on rule learning: III. Analyses and theoretical direction. Acta Psychologica, 1969, 29, 205-227. (b)

[47] Scandura, J.M., Barksdale, J., Dumin, J., & McGee, R. An unexpected relationship between failure and subsequent mathematics learning. Psychology in the Schools, 1969, 6, 379-381.

[50]Scandura, J.M. The role of rules in behavior: Toward an operational definition of what (rule) is learned. Psychological Review, 1970, 77, 516-533.

[53]Scandura, J.M. A theory of mathematical knowledge: Can rules account for creative behavior? Journal of Research in Mathematics Education, 1971, 2, 183-196. (a)

[54]Scandura, J.M. What is a rule? Journal of Educational Psychology, 1972, 63, 179-185.

[55]Scandura, J.M. Deterministic theorizing in structural learning: Three levels of empiricism. Journal of Structural Learning, 1971, 3, 21-53. (b)

[62]Scandura, J.M. & Dumin J.H. Assessing behavior potential: Adequacy of basic theoretical assumptions. Journal of Structural Learning, 1978, 6, 3-47.

[67]Scandura, J.M. The role of higher-order rules in problem solving. Journal of Experimental Psychology, 1974, 120, 984-991.

[70]Scandura, J.M., Dumin, J.H., & Wulfeck, W.H., II. Higher-order rule characterization of heuristics for compass and straight-edge constructions in geometry. Artificial Intelligence, 1974, 5, 149-183.

[93]Scandura, J.M. A deterministic theory of teaching and learning. In H. Spada & W.F. Kempf (Eds.), Structural models of thinking and learning. Bern: Huber, 1977. (a)

[97]Scandura, J.M. A deterministic approach to research in instructional science. Educational Psychologist, 1977, 12, 118-127. (b)

[102] Scandura, J.M. A. Discussion of selected issues relevant to structural learning. B. Comments on "Who is the learner." In J. M. Scandura & C.J. Brainerd (Eds.), Structural/process theories of complex human behavior. Leydon, The Netherlands: Sijthoff, 1978.

[108] Scandura, J.M. Structural approach to empirical testing: A contrast with normative methods in cognitive psychology. Journal of Structural Learning, 1978,6, 89-96. (a)

[112] Scandura, J.M. Structural learning theory. In C.M. Reigeluth (Ed.)*Instructional design theories and models: an overview of their current status*. Hillsdale: Erlbaum, 1984, 215-245. (a)

[113] Scandura, J.M. Problem solving in schools and beyond: Transitions from the naive to the neophyte to the master. Educational Psychologist, 1981, 16, 139-150.

[119] Scandura, J.M. Structural (cognitive task) analysis: A Method for Analyzing Content. Part 1: Background and Empirical Research. Journal of Structural Learning, 1982, 7, 101-114.

[130] Scandura, J.M. Structural Analysis. Part II: Toward precision, objectivity and systematization. Journal of Structural Learning, 1984, 8, 1-28. (a)

[141] Scandura, J.M., Stone, DC, & Scandura, A .B. "An intelligent RuleTutor CBI system for diagnostic testing and instruction". Journal of Structural Learning, 1986, 9, 15-61.

[146] Scandura, J.M. A structured approach to intelligent tutoring. Chapter 14 in D.H. Jonassen (Ed.) Instructional design for microcomputer software. Hillsdale, N.J.: Erlbaum, 1987, 347-379. (a)

[147] Scandura, J.M. A cognitive approach to software development: The PRODOC environment and associated methodology. Journal of Pascal, Ada & Modula-2, 6, 1987 (Sept./Oct.), pp. 10-25. (b)

[148] Scandura, J.M.. & Scandura, A.M. The intelligent RuleTutor: a structured approach to intelligent tutoring - Phase II. Journal of Structural Learning, 1987, 9, 195-259.

[156] Scandura, J.M. Cognitive technology and the PRODOC re/NuSys Workbench™: a technical overview. Journal of Structural Learning and Intelligent Systems, 1992, 11, pp. 89-126.

BEST COPY AVAILABLE

[159] Scandura, J.M. A cognitive approach to software development: The PRODOC environment and associated methodology. In D. Partridge (Ed.): Artificial Inteligence and Software Engineering. Norwood, NJ: Ablex Pub., 1991, Chapter 5, pp. 115-
[166] Scandura, J.M. Role of instructional theory in authoring effective and efficient learning technologies. Computers in Human Behavior, 1996, 12, 2, 313-328.

[162] Scandura, J.M. Automating renewal and conversion of legacy code into ada: A cognitive approach. IEEE Computer, 1994, April, 55-61.
[167] Scandura, J.M. Cognitive analysis, design and programming: next generation OO paradigm. Journal of Structural Learning and Intelligent Systems, 1997, 13, 1, 25-52.
[169] Scandura, J.M. & Scandura, A.B. Improving RAM in large software system development and maintenance. Journal of Structural Learning and Intelligent Systems, 1999, 13 (3-4) 227-294.
[B-2] Scandura, J.M., Durnin, J.H., Ehrenpreis, W., et al An algorithmic approach to mathematics: Concrete behavioral foundations. New York: Harper & Row, 1971.
[B-3] Scandura, J.M. Structural learning I: Theory and research London/New York: Gordon & Breach Science Publishers, 1973.
[B-6] Scandura, J.M. Problem Solving: a Structural / Process Approach with Instructional Implications. New York: Academic Press, 1977. (v)
[B-7] Scandura, J.M. & Brainerd, C.J. (Eds.). Structural/process models of complex human behavior. Alphen ann den Ryn, The Netherlands: Sitjthoff & Noordhoff, 1978. (b)
[B-8] Scandura, J.M. & Scandura, A.B. Structural Learning and Concrete Operations: An Approach to Piagetian Conservation. Praeger Sci. Pub., N.Y., 1980.
[P-3] Scandura, J.M. U.S. Patent (Formally Approved November, 2000), *Automated Methods for Building and Maintaining Software Based on Intuitive (Cognitive) and Efficient Methods for Verifying that Systems are Internally Consistent and Correct Relative to their Specifications.* In press.
Scandura, J.M. & Releucke, W. *Some Statistical Concerns in Structural Learning.* Chapter 10 in J.M. Scandura. Problem Solving: a Structural / Process Approach with Instructional Implications. New York: Academic Press, 1977.
Voorhies, D. & Scandura, J.M. *Determination of Memory Load in Information Processing.* Chapter 7 in J.M. Scandura. Problem Solving: a Structural / Process Approach with Instructional Implications. New York: Academic Press, 1977.
Wulfeck, W.H. & Scandura, J.M. *Theory of Adaptive Instruction with Application to Sequencing in Teaching Problem Solving.* Chapter 14 in J.M. Scandura. Problem Solving: a Structural / Process Approach with Instructional Implications. New York: Academic Press, 1977.

Appendix A

Historical Overview of Development and Applications of Structural Learning Theory

This appendix traces key ideas and research results leading to the SLT as it exists at the start of new millenium. Due to space limitations, and because its development was largely self-contained, this summary makes only passing reference to other research. As shown in Figure A, most of this early work centered on theoretical and empirical research on cognition and instruction. An initial set of results demonstrated the critical importance of prior learning in problem solving instruction (e.g., 2, 3, 12, 13, 14). By the mid-1960s, the work branched in three parallel directions.

1. The goal of "Psycho-mathematics" was to find better ways of integrating mathematics and psychology. At that time, experimental psychologists gave short shift to content (limiting concern in experimental research to ambiguous variables like "difficulty"). Mathematicians almost universally felt that psychology had little to say about their subject. By analogy to psycho-linguistics, the subject of concern was dubbed "psycho-mathematics" (e.g., 18, B-1).

2. Research in experimental and mathematical psychology (e.g., 11, 19) reinforced the fact that behavior is the basic observable in instructional science. Behavioral scientists do not have license to "look inside". Even if possible, it would be very difficult to extract useful information about the kinds of complex learning with which instructional scientists are routinely concerned.

While the prevailing view was that learning consisted of complexes of S-R associations (e.g., Gagne, 1964), SLT research adopted the *rule* as the basic unit of analysis. Simpler forms (e.g., concepts and S-R associations) were represented as special cases (e.g., 30, 43, 50, 53, 54). Experiments on rule learning (e.g., 44, 45) ranged from demonstrating the efficacy of rules in explaining simpler kinds of behavior to the importance of higher order rules in explaining problem solving behavior and "far transfer" from one domain to another (e.g., 38).

Research on one-trial learning and one-trial transfer also led to a discovery in assessing behavior potential. A single well-chosen test item was found sufficient for assessing what rule was learned with sufficient precision to enable deterministic predictions about behavior on individual items. As few as one test item was sufficient for this purpose (e.g., 30, 50, 62, 63).

3. These results motivated the first serious attempt to pull the ideas together in a coherent Structural Learning Theory (slt) at the Second International Structural Learning Conference in 1970 (55). Three levels of empiricism (competence, "memory-free" cognition and memory constrained cognition) were defined, each enabling deterministic as opposed to probabilistic predictors. This work was shortly followed by *Structural Learning: Theory and Research* (B-3), and other publications (B-4, B-5, B-7) offering a more detailed formulation and a substantial amount of supporting research.

By the mid-1970s, psychology turned its attention to cognition in mass. A whole spectrum of domain specific cognitive theories (intermixing content and cognitive issues) emerged within a short time period. Semantic networks and production systems were widely used in modeling knowledge. Little attention, however, was given to structural (cognitive task) analysis, cognitive universals (e.g., 113), knowledge assessment or ironically in some cases behavior itself. The latter was especially apparent in constructionist theories. In contrast, work on structural analysis, for example, gave special attention to identifying higher order knowledge and its role in problem solving (e.g., 70, 80, B-2, 119, 130, 131) and

BEST COPY AVAILABLE

automatization (B-8). Most cognitive research at the time was concerned with more general distinctions: domain-specific vs. domain-independent, declarative vs. procedural knowledge, etc. Increasing attention also was given to instructional implications (e.g., 93, 97, 109, 114), eventually realized in a variety of computer based instructional systems. Authoring systems developed in the late 1970s (S-5, S-6, 115) were used to develop almost a thousand different titles, some still being sold today even though they run only on the long obsolete Apple II computer. Subsequent work led to intelligent tutoring systems (e.g., S-4, 141, 146, 148), and more surprisingly to software engineering.

What began as a short digression immersed us in Computer Aided Software Engineering (CASE) for over a decade (e.g., 142, 154, 156, 159, 161, 162, 164, 167, 168,169). PRODOC and its successor Flexsys (S-9) made the need for dramatically increased rigor and precision increasingly clear. It took almost 15 years of intensive software R&D to provide a proper technical foundation (Softbuilder & its AutoBuilder extension) for appropriate implementation of fundamental structural learning concepts. Although AutoBuilder (S-10) focuses on software, it is in fact a natural evolution of earlier work in structural analysis. It offers both a new programming paradigm and a technical foundation on which to build general-purpose intelligent tutoring systems.

The long divergence into software engineering had a major impact on the SLT, in particular on the language used in its exposition and its breadth, cohesiveness and precision. In its present form, SLT and accompanying technologies open exciting new opportunities for research, including a systematic approach to the development of instructional and other intelligent systems.

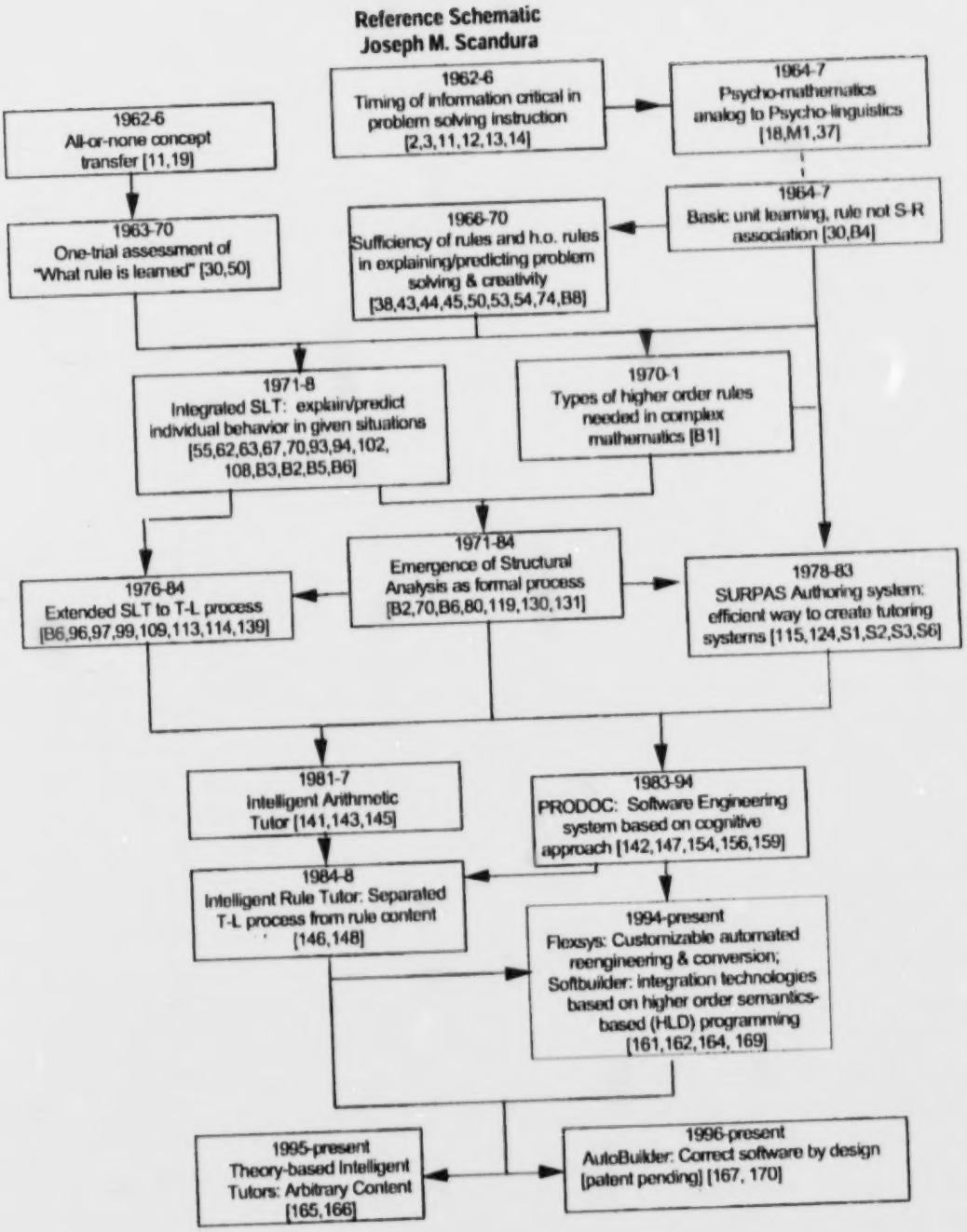
Reference Schematic
Joseph M. Scandura
1962-6
All-or-none concept transfer [11,19]
1962-6
Timing of information critical in problem solving instruction [2,3,11,12,13,14]
1964-7
Psycho-mathematics analog to Psycho-linguistics [18,M1,37]
1964-7
Basic unit learning, rule not S-R association [30,B4]
1963-70
One-trial assessment of "What rule is learned" [30,50]
1966-70
Sufficiency of rules and h.o. rules in explaining/predicting problem solving & creativity [38,43,44,45,50,53,54,74,B8]
1971-8
Integrated SLT: explain/predict individual behavior in given situations [55,62,63,67,70,93,94,102, 108,B3,B2,B5,B6]
1970-1
Types of higher order rules needed in complex mathematics [B1]
1971-84
Emergence of Structural Analysis as formal process [B2,70,B6,80,119,130,131]
1976-84
Extended SLT to T-L process [B6,96,97,99,109,113,114,139]
1978-83
SURPAS Authoring system: efficient way to create tutoring systems [115,124,S1,S2,S3,S6]
1981-7
Intelligent Arithmetic Tutor [141,143,145]
1983-94
PRODOC: Software Engineering system based on cognitive approach [142,147,154,156,159]
1984-8
Intelligent Rule Tutor: Separated T-L process from rule content [146,148]
1994-present
Flexsys: Customizable automated reengineering & conversion; Softbuilder: integration technologies based on higher order semantics-based (HLD) programming [161,162,164, 169]
1995-present
Theory-based Intelligent Tutors: Arbitrary Content [165,166]
1996-present
AutoBuilder: Correct software by design [patent pending] [167, 170]

Figure A: Reference Schematic summarizing important developments in Structural Learning Theory.

BEST COPY AVAILABLE

Additional Scandura References (Unique to Appendix A)

11. Scandura, J. M. Familiarization and one-trial learning: A positive interaction. Psychological Reports, 1965, 17, 27-38.
12. Scandura, J. M. Problem solving and prior learning. Journal of Experimental Education, 1966, 34, 7-11.
13. Scandura, J. M. Algorithm learning and problem solving. Journal of Experimental Education, 1966, 34, 1-6.
14. Scandura, J. M. Prior learning, presentation order, and prerequisite practice in problem solving. Journal of Experimental Education, 1966, 34, 12-18.
18. Scandura, J. M. Precision in research on mathematics learning: The emerging field of psycho-mathematics. Journal of Research in Scientific Teaching, 1966, 4, 253-274.
19. Greeno, J. G. & Scandura, J. M. All-or-none transfer based on verbally mediated concepts. Journal of Mathematical Psychology, 1966,3, 388-411.
30. Scandura, J.M. The basic unit in meaningful learning-association or principle? The Proceedings, APA 1967. (Also in The School Review,[1967, 75, 329-341.)
74. Scandura, J.M. On higher-order rules. Educational Psychologist, 1973, 10, 159-160.
80. Scandura, J.M., Wulfeck, W. H., II, Dumin, J. H., & Ehrenpreis, W. Diagnosis and instruction of higher-order rules for solving geometry construction problems. In 1974 Proceedings; see B-5. (Also in Problem Solving; see B-6.)
96. Scandura, J.M. The architecture of bricklaying (A review of B. Pask's Conversation, cognition, and learning. Amsterdam: Elsevier, 1975). Contemporary Psychology, 1976, 21, 630-631.
99. Scandura, J.M. Cybernetic control of adaptive algorithms in instruction (a review of L.N. Landa's Instructional regulation and control (cybernetics, algorithmization, and heuristics in education.) National Society of Programmed Instruction, 1977, 16, 14-15.
102. Scandura, J.M. A. Discussion of selected issues relevant to structural learning. B. Comments on "Who is the learner." In J. M. Scandura & C.J. Brainerd (Eds.), Structural/process theories of complex human behavior. Leydon, The Netherlands: Sijthoff, 1978.
108. Scandura, J.M. Structural approach to empirical testing: A contrast with normative methods in cognitive psychology. Journal of Structural Learning, 1978,6, 89-96.
109. Scandura, J.M. Theoretical foundations of instruction: A systems alternative to cognitive psychology. Journal of Structural Learning, 1980, 6, 347-394. (Also appears as Appendix in B-8)
114. Scandura, J.M. Cognitive instructional psychology: System requirements and research methodology. Journal of Computer Based Instruction, 1984, 11, 32-41.
115. Scandura, J.M. Microcomputer systems for authoring, diagnosis and instruction in rule-based subject matter. Educational Technology, 1981, 13-19. Also published in proceedings, International Congress of Applied Systems and Cybernetics, Acapulco, Mexico, December 12-15, 1980.
124. Scandura, J.M. Computer-Based Instruction: Specific Goals or Intangibles. AEDS Monitor, 1982.
131. Scandura, J.M. Structural Analysis. Part III: Validity and reliability. Journal of Structural Learning, 1984, 8, 173-193.
142. Scandura, J.M. PRODOC: The PROfessional self-DOCumenting programming productivity environment. Journal of Structural Learning, 1986, 9, 101-105.
143. Scandura, J.M. "An ICBI RuleTutor for diagnosis and prescription in arithmetic." FOCUS (Journal of Research Council for Diagnostic and Prescriptive Mathematics). Special Issue, 1987, 9, 53-67.
145. Scandura, J. M. Comments on the arithmetic RuleTutor. Educational Technology, 1986.
154. Scandura, J.M. Cognitive approach to systems engineering and re-engineering: integrating new designs with old systems. Journal of Software Maintenance, 1990, 2, 145-156.
161. Scandura, J.M. Automating renewal and conversion of legacy code. Software Engineering Strategies. NY: Auerbach Publications, 1994, March/April, pp. 31-43. (Reprinted in Handbook of Systems Management, Development and Support, Auerbach, 1995. Similar version in Scuola estiva: Engineering of existing software. Bari, Italy: Dipartimento di Informatica Universita degli Studi di Bari, 1994, pp. 179-192.)
164. Scandura, J.M. Cognitive analysis, design and programming: generalization of the object oriented paradigm. Valley Forge, PA: Proceedings of the National Ada Conference, March 1995.
165. Scandura, J.M. Theoretical foundations of instruction: past, present and future. Journal of Structural Learning, 1995, 12, 231-243.
168. Scandura, J.M. A cognitive approach to reengineering. Crosstalk, The Journal of Defense Software Engineering, June 1997, 10, 6, 26-31.

BOOKS AND MONOGRAPHS

M-1. Scandura, J.M. (Ed.). Research in mathematics education. Washington, DC: NCTM, 1967.
B-1. Scandura, J.M. Mathematics: Concrete behavioral foundations. New York: Harper & Row, 1971.
B-3. Scandura, J.M. Structural learning I: Theory and research London/New York: Gordon & Breach Science Publishers, 1973.
B-6. Scandura, J.M. Problem Solving: a Structural / Process Approach with Instructional Implications. New York: Academic Press, 1977.
B-7. Scandura, J.M. & Brainerd, C.J. (Eds.). Structural/process models of complex human behavior. Alphen ann den Ryn, The Netherlands: Sijthoff & Noordhoff, 1978.
B-8. Scandura, J.M. & Scandura, A.B. Structural Learning and Concrete Operations: An Approach to Piagetian Conservation. Praeger Sci. Pub., N.Y., 1980.

MAJOR COMPUTER-BASED SYSTEMS

MicroSystem80 Tutorial Series (published by Borg-Warner Educational Systems):

S-1. Critical Reading, Fall 1980 (8 disk system)

S-2. College Board Preparation, Winter 1980 (12 disk system)
S-3. Language Arts - Word Structure, Fall 1982 (8 disk system)

MicroTutor II Tutorial Series (published by Queue, Inc.):

S-4. Tutorial Arithmetic, Fall 1982 (12 disk system for addition, subtraction, multiplication, division).
S-5. EZ Author & EZ Author Plus Authoring systems, Spring 1983.
S-6. SURPAS Authoring system with "intelligent" interface, Spring 1983.
S-7. Mosaic Tutorials (over 200 titles covering areas from pre-school perceptual skills to college preparatory science and mathematics).

Software Engineering (published by Scandura.com)

S-8. PRODOC re/NuSys Workbench, 1990.
S-9. Flexsys, 1995-present.
S-10. Softbuilder, 1999-present
S-11. AutoBuilder, 1995-present

PATENTS

P-1. U.S. Patent No: 5,262,761 *Displaying hierarchical tree-like designs in windows*, Scandura et al, Nov. 1993.
P-2. U.S. Patent No. 6,061,513, *Automated Method for Constructing Language Specific Systems for Reverse Engineering Source Code into Abstract Syntax Trees with Attributes in a Form that can More Easily be Displayed, Understood and/or Modified*. May 9, 2000.
P-4 U.S. Patent (pending). Method for Developing and Maintaining Distributed Systems via Plug and Play. May, 2000.

Appendix B

RELATIONSHIPS TO FORMAL WORK IN COMPUTER SCIENCE[11]

It is interesting to contrast this strictly cognitive approach to competence with more formal research in computer science. In logic programming, for example, propositions in a (e.g., knowledge based) theory are to be proved (or disproved) based on given sets of assumptions together with some set of inference rules. A good deal of effort has and is going into constructing formal languages for representing assumptions and formulating inference rules which make it possible to deal with broad ranges of problem domains in an efficient manner. As every mathematician knows, Godel's (1931) incompleteness theorem shows that no one set of assumptions or inference rules will suffice to prove (or disprove) every well-formulated statement concerning the non-trivial models with which they work.[12] Consequently, most current research on automation involves human interaction, in which automated methods are being supplemented with human input.

The above cognitive approach finesses this problem by reformulating the problem. Rather than seeking fundamental sets of assumptions and inference rules, from which all conjectures are to be drawn, our approach might be characterized as starting with arbitrarily given sets of conjectures and constructing those sets of assumptions and inference rules which offer a sufficiently adequate account of the desired behavior. Human input in this case is gathered together at the beginning.

For illustrative purposes, consider the well-known KIV system for verifying program correctness. KIV was developed over many years at Saarbrucken, Karlsruhe and Ulm universities in Germany, and represents an important achievement. Its biggest success was to prove correctness of a Prolog compiler. KIV is written in LISP, and based on First-order Algebraic Specifications, a Pascal like language, for constructing specifications and dynamic logic to prove correctness of imperative programs with respect to given specifications. This language is used in KIV to create specifications and correctness assumptions to be used in program verification.

The user is free to verify program correctness, either interactively as the process proceeds or after a program has been completed. With smaller simpler constructions KIV is often able to construct verifications on its own. In most cases, however, a high degree of highly skilled interactivity is required of the user (e.g., users must specify loop-invariants, which can be quite difficult in practice). Successful use of KIV in practice almost invariably requires heavy involvement of KIV's developers, who are logicians as well as programmers. In generating its proofs, KIV uses almost 80 inference rules, about 10 for first order predicate logic and the others for various programming constructs. More important here, it also uses a large number of heuristics to resolve conflicts which arise in situations where more than one inference rule may be applied. These heuristics were added gradually as needed to resolve conflicts arising during KIV's development.

An SLT approach to the problem would differ in some important ways. First, the representation of both specifications and programs would be based on epistemological

---

[11] I would like to thank Uli Furbach, Peter Baumgartner and Ingo Dahn for their helpful comments on these remarks. I am indebted to Peter also for giving me a personal demonstration of the KIV system. As a usual disclaimer, of course, it should be emphasized that I am fully responsible for these remarks and any errors therein.

[12] What may be simple for mathematicians, however, can pose complex issues for the logician. Quantification over predicates, for examples, introduces complications that do not exist in first order logic (involving quantification only over variables).

considerations. Suffice it to say here that both specifications and designs are represented in a simple High Level Design (HLD) language that is well suited to representing cognitive processes. All processes are represented in the form *operation (in-par1, ..: in-out-Par1, ... ; out-par1, ...)*. Parameters represent arbitrarily complex structures but are otherwise typeless. A major advantage of HLD is that it is easily learned not only by programmers but by essentially anyone with expertise in the problem domain.

Second, the method of SA would be used to facilitate identification of the assumptions, inference rules and heuristics associated with some given domain of specifications. As per our analysis, mastery of the semantics of assumptions and conjectures is an essential prerequisite to any SA. Identification of lower and higher rules of competence is only meaningful after the effective inputs and outputs have been identified. In this context, one has the option of starting with a small domain, and gradually enlarging it, or otherwise.

Third, assumptions, inference rules and heuristics would be treated equally in constructing proofs. The SLT control mechanism would be used in determining which rule to use under which circumstances. The search at each level (of goal switching) is strictly linear with respect to the number of rules (assumptions, inference rules and heuristics).

Fourth, conflict resolution would be based on heuristics (i.e., higher order selection rules) introduced as needed during SA.

As mentioned above, most working in the area of automatic theorem proving (or proving correctness of programs) believe some degree of human interaction will be crucial. Recent research by Dahn is suggestive of how this might be done within a SLT perspective.

Structural decomposition is a basic method in the discovery of mathematical proofs. Thus, an intermediate step C may be used to prove that a theorem B is a consequence of some set A of assumptions. This decomposes the problem of proving A -> B into two problems: proving that A implies C and that A together with C implies B. This way of decomposing a proof problem is known as a "cut" in the mathematical theory of logical calculi. It is similar to the method of sequential decomposition in software design.[13] Indeed, the same general approach is commonly used in all problem solving.

Another way of reducing proof problems is to introduce subproofs that concentrate on the solution of specific subproblems. Case distinctions can be seen as a particular instance of this method. This method of identifying subcategories also is common in all problem solving. Both methods from an SLT perspective correspond to higher order rules, which operate on given problems and breaks them down in to subproblems.[14]

---

[13] According to Dahn, formalized first order logic examples have been discovered which have short proofs using cuts but only very long cut-free proofs (in a sense to be made precise). Mathematicians have developed higher order heuristic rules to guess the intermediate formula C (also see Scandura & Durnin, Chap. 4 in B-6).

[14] Dahn points out some of the subtleties that arise in mathematics:. To prove the triangle inequality |a+b| =< |a|+|b| for real numbers a and b, one might try to find two properties C_1,C_2 such that "C_1 or C_2" holds and that each of these conditions implies the desired inequality. This will yield three subproblems that can be solved independently. The choice of C_1,C_2 in our example can guided by several heuristic rules. Experience with linear orderings suggests that C_1 may be of the form X =< Y while C_2 is X >= Y.

Definition of the absolute value function suggests that X (or Y) might be 0. Hence, the beginner might try to distinguish the cases a =< 0 and a >= 0 and similarly for b which leads to four cases. The expert , however, may make the case distinction more simply. In our example one case distinction (a+b >= 0 or a+b =< 0) is sufficient to solve the problem.

Problem decomposition helps to identify essential aspects. When a proof is structured into subproofs, each can be attacked individually. Similarly, structuring of the knowledge base (axiom system) makes it easier to identify knowledge that is potentially useful to solve a particular problem.

The role of structure in knowledge processing is indispensable when it comes to automation. In seeking a solution, indiscriminant use of all available knowledge leads to a combinatorial explosion in the size of the search space. To cope with this problem, Dahn and Wolf[15] have developed a calculus that allows humans to reduce proof problems to a level of detail that can be handled by automated theorem provers.[16] [17] In this case, human input plays the role of higher order knowledge with theorem provers supplying the necessary lower order knowledge.

The main contribution of structural analysis would be in helping to make that higher order knowledge explicit so that it might be taught to others or even incorporated into an automated system.

In making these proposals, I hasten to add that while I have some (limited) sense of the field, I am not a logician. There is much I do not know about such currently active fields of research as logic programming and theorem proving -- or any primarily formal approach to computer science for that matter. There may be other work that directly relates to that described herein. If so, I can only: a) reemphasize that the work described herein is a natural outgrowth of 30-year-old work on cognition and b) hope that my work may be of some help in motivating more formal developments.

---

[15] I. Dahn, A. Wolf: A calculus supporting structured proofs, J. Inf. Proc. Cyb. 5/6 1994, 261-276.

[16] I. Dahn, J. Denzinger: Cooperating Theorem Provers. In Automated deduction: A basis for application (W. Bibel, P. H. Schmitt eds.), Vol. II, 383-416, Kluwer 1998.

[17] Identification of knowledge structures that are potentially useful in reaching a given goal is a largely unsolved problem. Moreover, techniques must be developed to make efficient use of this knowledge. In mathematics this has been intensively studied for properties of equality and to a certain extent for associatively and commutativity of algebraic operators. The implementation of meta-rules concerning the use of these properties by W. McCune has been essential for the solution of the famous Robbins conjecture, open for more than 60 years, by the automated theorem prover EQP [I. Dahn, Robbins Algebras are Boolean - A Revision of McCunes Computer Generated Solution of Robbins Problem, J. Algebra vol. 208 (1998), 526-532, 1998].

U.S. Department of Education
Office of Educational Research and Improvement (OERI)
National Library of Education (NLE)
Educational Resources Information Center (ERIC)

ERIC Educational Resources Information Center

REPRODUCTION RELEASE

(Specific Document)

TM034993

I. DOCUMENT IDENTIFICATION:

http://www.scandura.com/Articles/SLT%20Status-Perspectives.PDF

Title: Structural Learning Theory in the 21st Century

Author(s): Joseph M. Scandura

Corporate Source: MERGE Research Institute
1249 Greentree Lane, Narberth PA 19072

Publication Date: Nov 2001

II. REPRODUCTION RELEASE:

In order to disseminate as widely as possible timely and significant materials of interest to the educational community, documents announced in the monthly abstract journal of the ERIC system, *Resources in Education* (RIE), are usually made available to users in microfiche, reproduced paper copy, and electronic media, and sold through the ERIC Document Reproduction Service (EDRS). Credit is given to the source of each document, and, if reproduction release is granted, one of the following notices is affixed to the document.

If permission is granted to reproduce and disseminate the identified document, please CHECK ONE of the following three options and sign at the bottom of the page.

The sample sticker shown below will be affixed to all Level 1 documents

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL HAS BEEN GRANTED BY

Sample

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

1

Level 1

☐

Check here for Level 1 release, permitting reproduction and dissemination in microfiche or other ERIC archival media (e.g., electronic) *and* paper copy.

The sample sticker shown below will be affixed to all Level 2A documents

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL IN MICROFICHE, AND IN ELECTRONIC MEDIA FOR ERIC COLLECTION SUBSCRIBERS ONLY, HAS BEEN GRANTED BY

Sample

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

2A

Level 2A

☒

Check here for Level 2A release, permitting reproduction and dissemination in microfiche and in electronic media for ERIC archival collection subscribers only

The sample sticker shown below will be affixed to all Level 2B documents

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL IN MICROFICHE ONLY HAS BEEN GRANTED BY

Sample

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

2B

Level 2B

☐

Check here for Level 2B release, permitting reproduction and dissemination in microfiche only

Documents will be processed as indicated provided reproduction quality permits.
If permission to reproduce is granted, but no box is checked, documents will be processed at Level 1.

I hereby grant to the Educational Resources Information Center (ERIC) nonexclusive permission to reproduce and disseminate this document as indicated above. Reproduction from the ERIC microfiche or electronic media by persons other than ERIC employees and its system contractors requires permission from the copyright holder. Exception is made for non-profit reproduction by libraries and other service agencies to satisfy information needs of educators in response to discrete inquiries.

Sign here, → please

Signature: ✓ Alice B. Scandura	Printed Name/Position/Title: Alice B. Scandura, Exec Dir.
Organization/Address: MERGE Research Institute, 1249 Greentree Lane, Narberth, PA 19072	Telephone: 610-664-1207 FAX:
	E-Mail Address: alice@scandura.com Date: 1/10/03

(Over)

III. DOCUMENT AVAILABILITY INFORMATION (FROM NON-ERIC SOURCE):

If permission to reproduce is not granted to ERIC, *or*, if you wish ERIC to cite the availability of the document from another source, please provide the following information regarding the availability of the document. (ERIC will not announce a document unless it is publicly available, and a dependable source can be specified. Contributors should also be aware that ERIC selection criteria are significantly more stringent for documents that cannot be made available through EDRS.)

Publisher/Distributor:
Address:
Price:

IV. REFERRAL OF ERIC TO COPYRIGHT/REPRODUCTION RIGHTS HOLDER:

If the right to grant this reproduction release is held by someone other than the addressee, please provide the appropriate name and address:

Name:
Address:

V. WHERE TO SEND THIS FORM:

Send this form to the following ERIC Clearinghouse:

ERIC CLEARINGHOUSE ON ASSESSMENT AND EVALUATION
UNIVERSITY OF MARYLAND
1129 SHRIVER LAB
COLLEGE PARK, MD 20742-5701
ATTN: ACQUISITIONS

However, if solicited by the ERIC Facility, or if making an unsolicited contribution to ERIC, return this form (and the document being contributed) to:

ERIC Processing and Reference Facility
4483-A Forbes Boulevard
Lanham, Maryland 20706

Telephone: 301-552-4200
Toll Free: 800-799-3742
FAX: 301-552-4700
e-mail: ericfac@inet.ed.gov
WWW: http://ericfacility.org

EFF-088 (Rev. 2/2001)